# 目录

[目录 1](#_Toc16524609)

[第4章 STM32F407VG-中断 2](#_Toc16524610)

[4.1 中断介绍 2](#_Toc16524611)

[4.2 Cortex-M4中断体系 2](#_Toc16524612)

[4.3 STM32F407VG中断介绍 4](#_Toc16524613)

[4.4 STM32F407VG片上外设中断 6](#_Toc16524614)

[4.4.1 STM32F407VG片上外设中断工作原理 6](#_Toc16524615)

[4.4.2 STM32F407VG片上外设中断—USART1的中断相关寄存器 7](#_Toc16524616)

[4.4.3 STM32F407VG片上外设中断软件程序 7](#_Toc16524617)

[4.5 STM32F407VG外部中断 9](#_Toc16524618)

[4.5.1 STM32F407VG外部中断简介 9](#_Toc16524619)

[4.5.2 STM32F407VG外部中断原理 10](#_Toc16524620)

[4.5.3 STM32F407VG外部中断相关寄存器 12](#_Toc16524621)

[4.5.4 STM32F407VG外部中断实验 17](#_Toc16524622)

[4.6 STM32F407VG软件中断 18](#_Toc16524623)

[4.6.1 STM32F407VG软件中断介绍 18](#_Toc16524624)

[4.6.2 STM32F407VG软件中断工作原理 18](#_Toc16524625)

[4.6.3 STM32F407VG软件中断实验 19](#_Toc16524626)

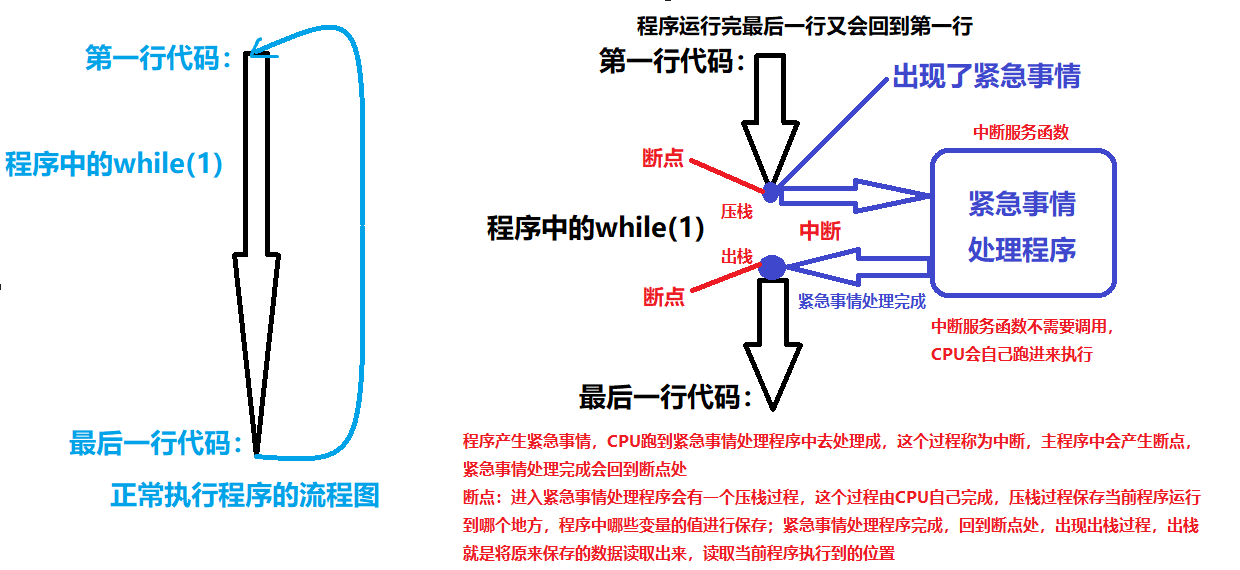
# STM32F407VG-中断

## 中断介绍

中断作用：用于处理程序中的紧急事情。

中断相关概念：

中断结构：需要清楚中断的执行过程。

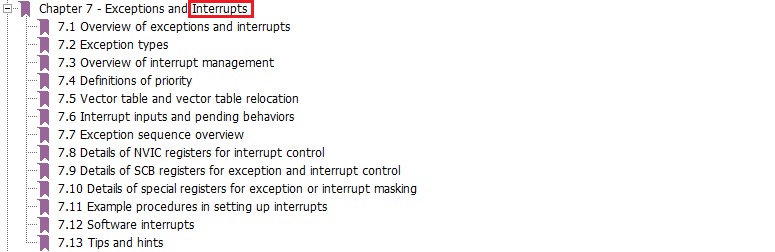


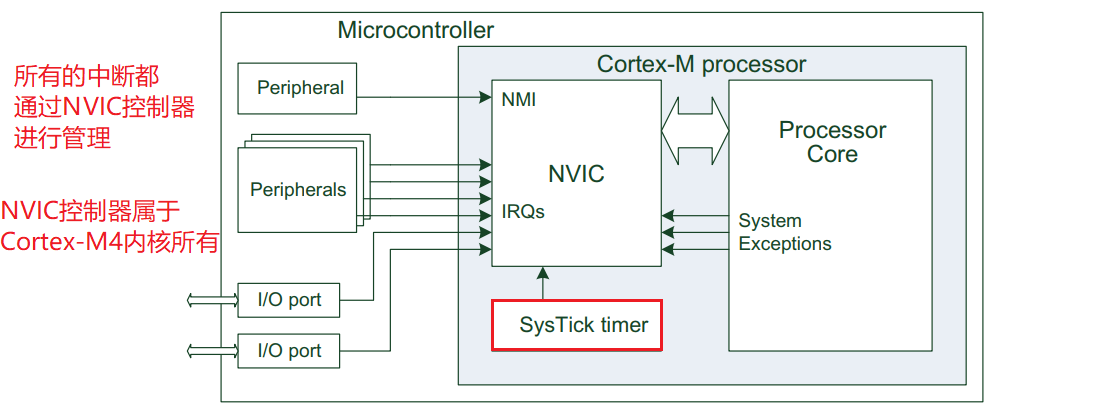
中断优先级：有很多紧急事情需要处理，对于不同的紧急事情处理优先程度不一致，有些事情处理需要很高优先级，有些事情可以慢一点。利用一个数值进行优先级区别，数值越小，优先级越高。对不同事情进行优先级排号，这个排号称为自然优先级，这个优先级由厂商指定。除了自然优先级还有工程师可以设置的优先级，这个优先级比自然优先级权力大。

中断嵌套：在执行中断的紧急处理程序时，又发生了更紧急的事情，又去处理了这个更紧急的事情。

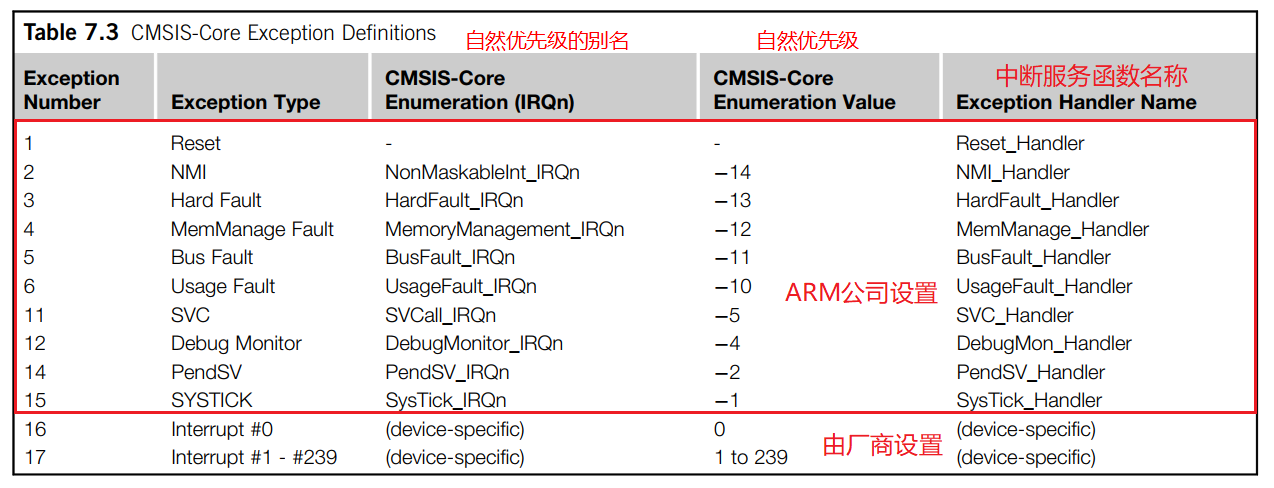
## Cortex-M4中断体系

需要查找内核资料。通过《Cortex M3与M4权威指南.pdf》手册进行查找。

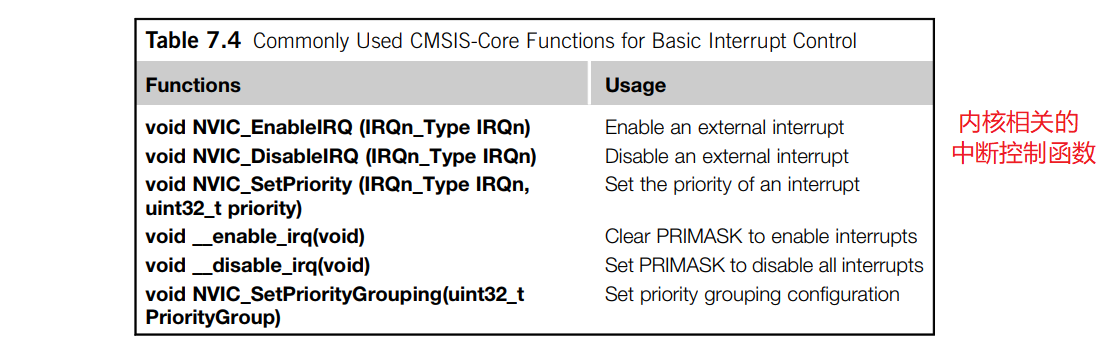




整个中断体系中，有ARM内核的系统级别的中断，有属于厂商设定的中断。



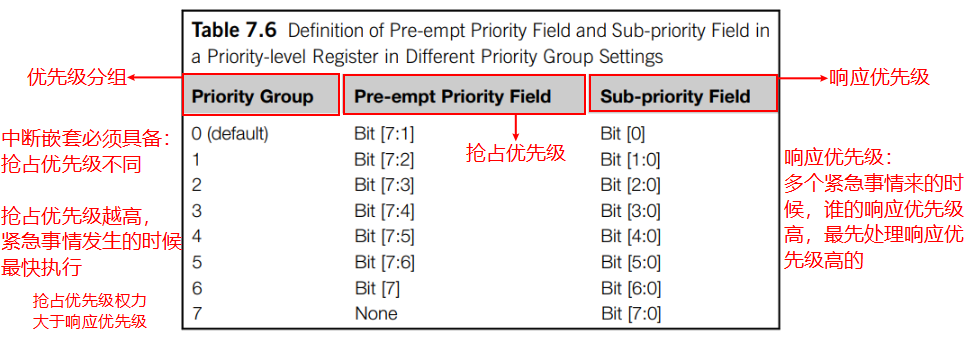
The reason for using a different number system in CMSIS-Core access functions is to allow slightly better efficiency in some of these API functions (e.g.,when setting up priority levels).



配置中断需要以下步骤：

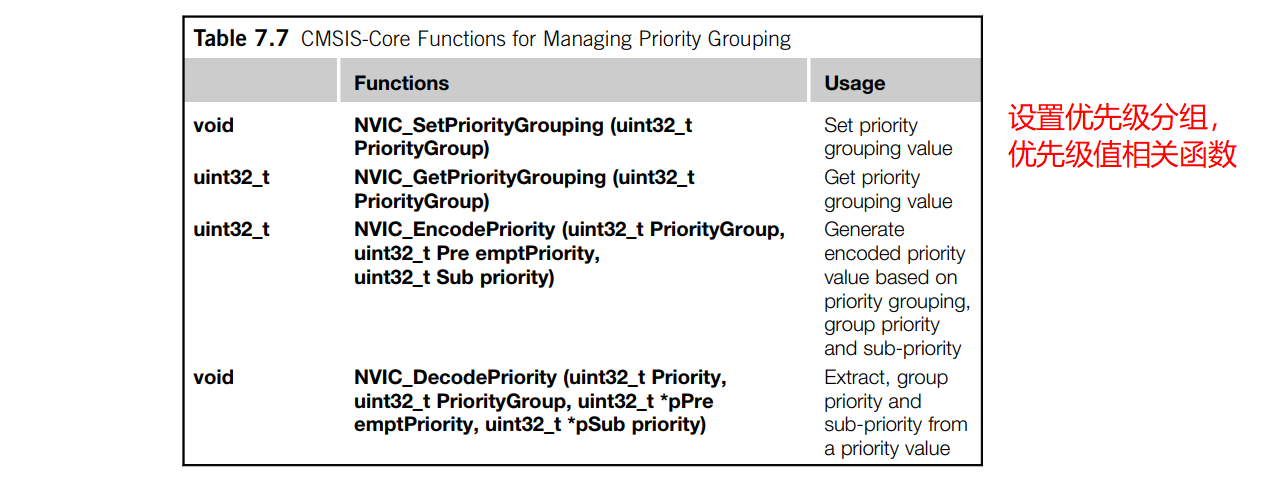
1. 设置优先级值（分组信息，抢占优先级，响应优先级）
2. 配置外设中断，外设中断的触发方式（配置外设的中断使能）
3. 配置NVIC控制器中断（配置NVIC控制器中断使能）

优先级分组表：



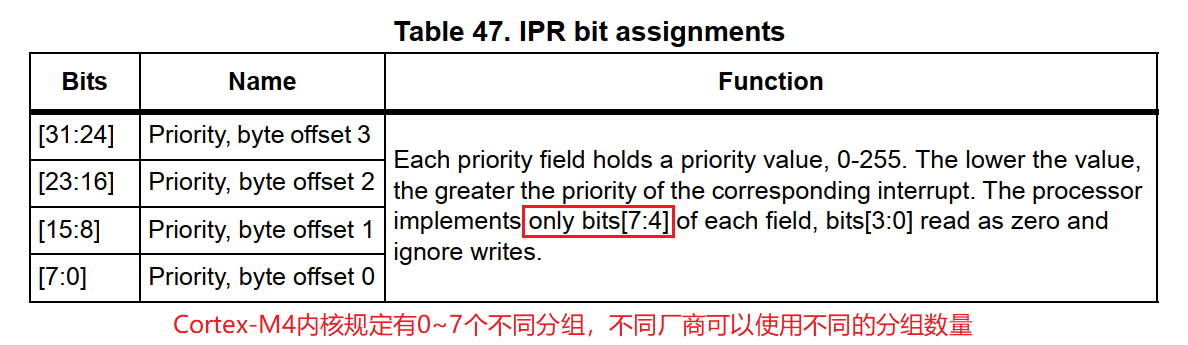
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优先级  分组 | 抢占优先级范围 | 抢占优先级设置的值 | 响应优先级范围 | 响应优先级设置的值 |
| 7 | 0 | 没有(0) | [7:0] | 0~255 |
| 6 | [7] | 0~1 | [6:0] | 0~127 |
| 5 | [7:6] | 0~3 | [5:0] | 0~63 |
| 4 | [7:5] | 0~7 | [4:0] | 0~31 |
| 3 | [7:4] | 0~15 | [3:0] | 0~15 |
| 2 | [7:3] | 0~31 | [2:0] | 0~7 |
| 1 | [7:2] | 0~63 | [1:0] | 0~3 |
| 0 | [7:1] | 0~127 | [0] | 0~1 |

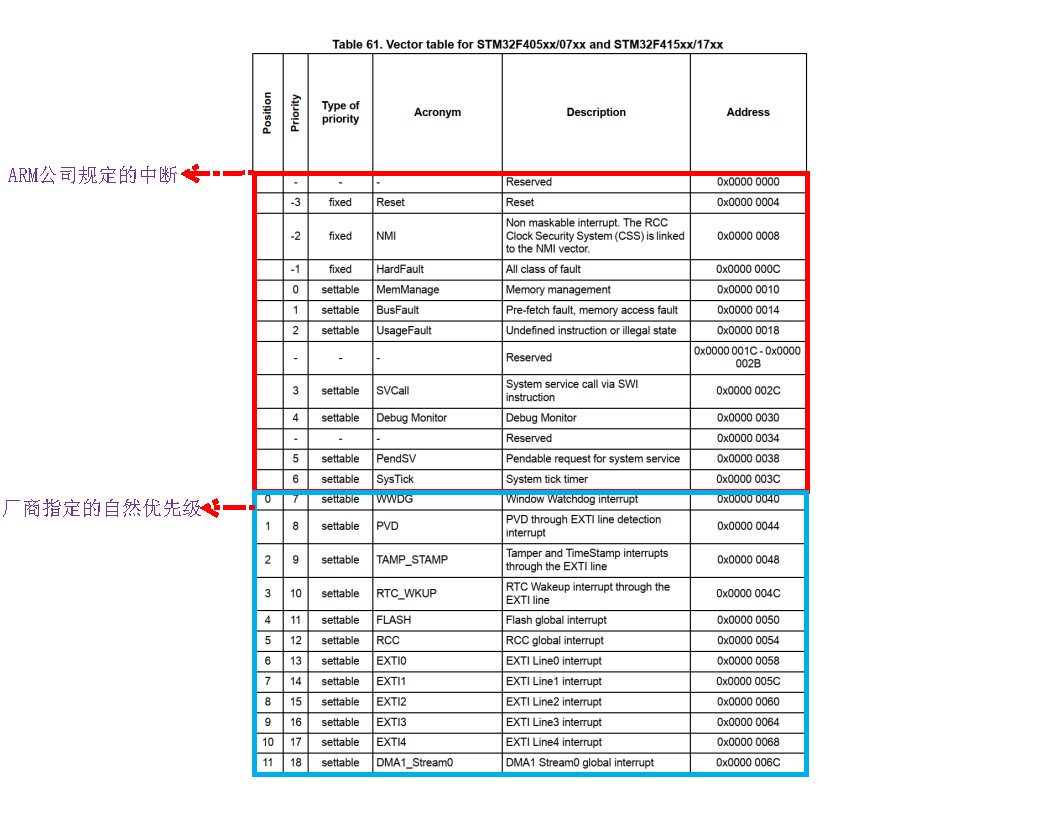
You can access the priority group setting, and encode/decode priority information using CMSIS-Core functions as shown in Table 7.7.



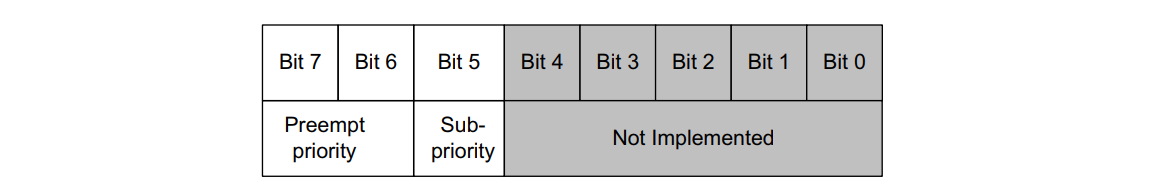
## STM32F407VG中断介绍

基于Cortex-M4内核架构的处理器，中断功能基于Cortex-M4内核的设置。



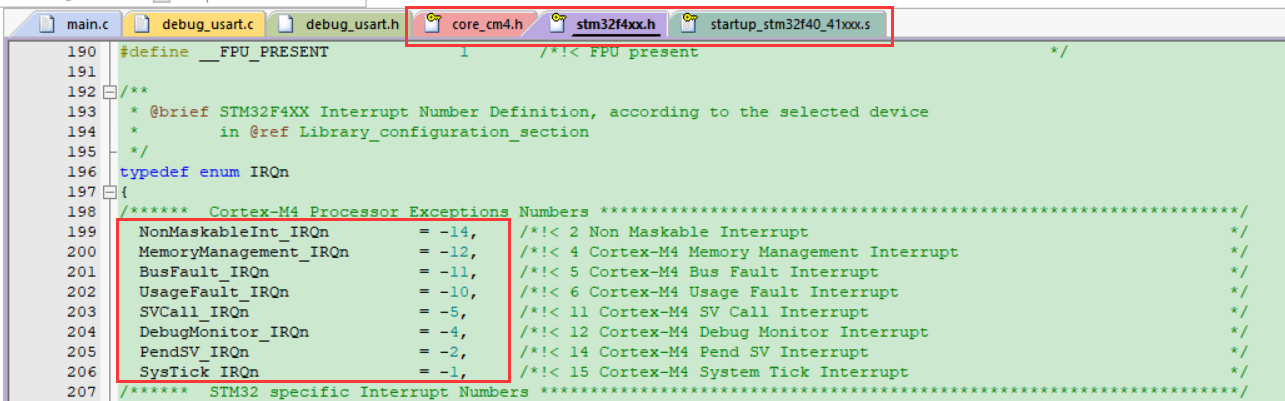


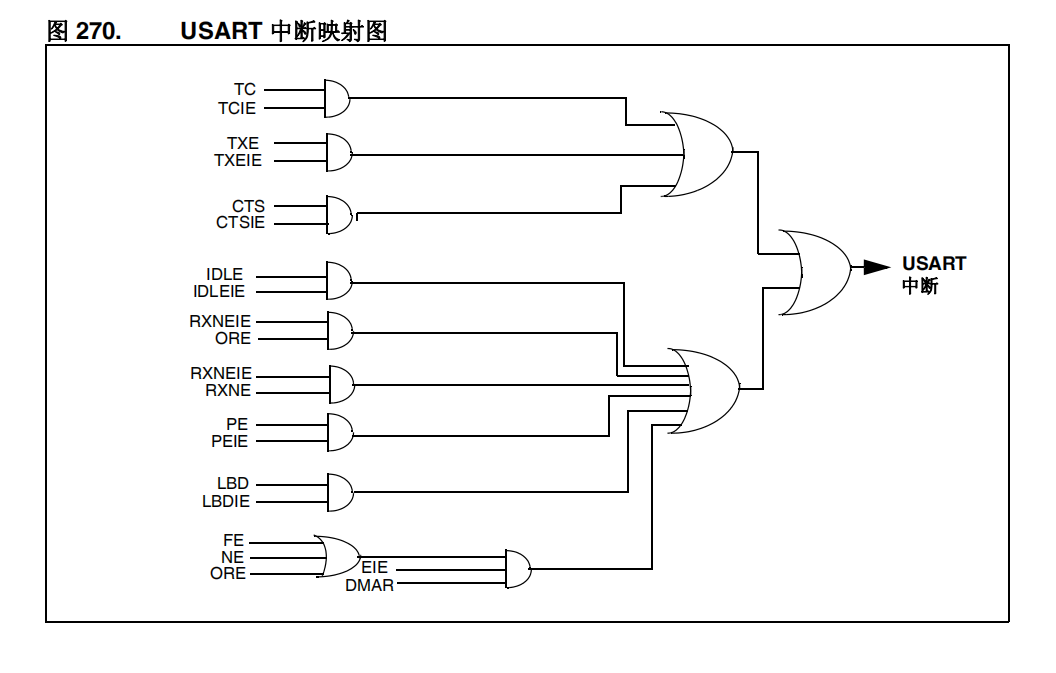
For example, if the width of the configuration registers is 3 (bit 7 to bit 5 areavailable) and priority group is set to 5, you can have four levels of group/pre-empt priority levels (bit 7 to bit 6), and inside each group/pre-empt level there are two levels of sub-priority (bit 5) (Figure 7.5)



当前芯片优先级设置位的宽度4个位，有效位处于[7:4]。需要根据内核分组去区分这些位该如何设置。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优先级  分组 | 抢占优先级范围 | 抢占优先级设置的值 | 响应优先级范围 | 响应优先级设置的值 |
| 7 | 0 | 没有(0) | [7:4] | 0~15 |
| 6 | [7] | 0~1 | [6:4] | 0~7 |
| 5 | [7:6] | 0~3 | [5:4] | 0~3 |
| 4 | [7:5] | 0~7 | [4] | 0~1 |
| 3 | [7:4] | 0~15 | 0 | 没有 |



厂商对于芯片内部的外设的自然优先级排布：需要查看STM32参考手册。

一个芯片中优先级分组只能是一种，分组值一旦确定不能修改。

按照ARM公司规定的一些设置函数编写一个简单示例：

|  |
| --- |
| 第一步：设置优先级  1.设置优先级分组  2.将分组、抢占优先级、响应优先级合成一个优先级值  3.将合成的优先级值设置到具体的中断中  第二步：设置外设中断使能  根据外设配置相关的寄存器进行设置  第三步：设置NVIC中断使能  调用ARM公司规定的NVIC中断使能函数  程序代码：  uint32\_t priority = 0; //保存合成的优先级值  NVIC\_SetPriorityGrouping(7 - 2);  priority = NVIC\_EncodePriority(7 - 2, 2, 2);  //NVIC\_SetPriority(37, priority); //以USART1为例//stm32f4xx.h的枚举中有  NVIC\_SetPriority(USART1\_IRQn, priority); //以USART1为例  //根据外设寄存器配置----需要掌握该外设相关的寄存器  NVIC\_EnableIRQ(USART1\_IRQn); //以USART1为例 |

## STM32F407VG片上外设中断

### STM32F407VG片上外设中断工作原理

片上外设中断部分配置：

|  |
| --- |
| 第一步：设置优先级  1.设置优先级分组  2.将分组、抢占优先级、响应优先级合成一个优先级值  3.将合成的优先级值设置到具体的中断中  第二步：设置片上外设中断使能  根据片上外设配置相关的寄存器进行设置  第三步：设置NVIC中断使能  调用ARM公司规定的NVIC中断使能函数 |

想要片上外设中断能够正常工作，必须满足该外设能够在查询方式下正常工作。

片上外设已经能够在查询方式下功能，只需加上片上外设中断部分的代码即可，就可以产生中断。

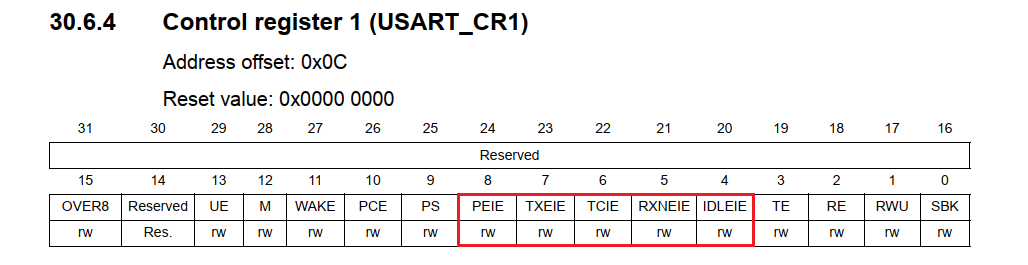
片上外设中断很多，该如何使用？

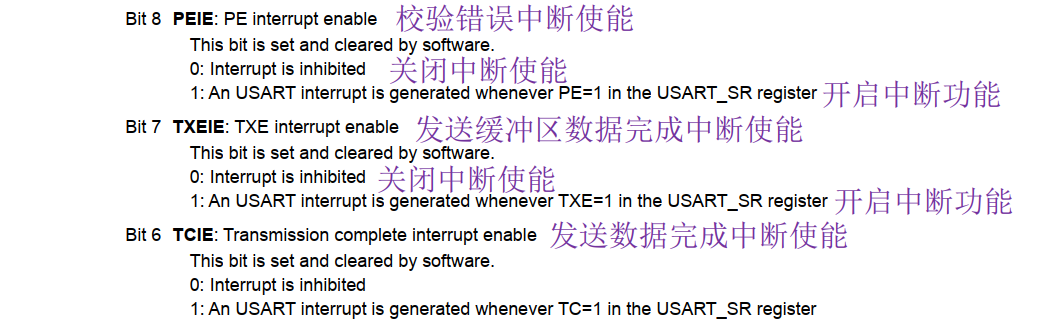
中断作用：处理紧急事情。什么算紧急事情？有紧急信号来的时候需要立马处理的事情属于紧急事情。紧急信号什么时候到达不知道？这种情况下必须使用中断。

以USART1的接收数据为例：

|  |
| --- |
| 需要配置USART1  学习USART1的中断相关寄存器  配置USART1的中断相关寄存器  编写USART1的中断服务函数  注意：在中断服务函数里面需要清除中断标志位。中断服务函数中的程序代码不能加等待。 |

### STM32F407VG片上外设中断—USART1的中断相关寄存器







### STM32F407VG片上外设中断软件程序

|  |
| --- |
| /\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数功能: 初始化USART1  \* 函数形参: None  \* 函数返回值: None  \* 备注: None  \* 作者: LeeSong  \* 时间: 2019-08-06  \* 修改作者: None  \* 修改时间: None  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  void debug\_usart\_init(void)  {  float usart\_div = 0;  u16 div\_mantissa = 0;  u16 div\_fraction = 0;    debug\_usart\_port\_init();    RCC->APB2ENR |= (0X1 << 4); //开启USART1的时钟  usart\_div = (float)84000000ul / (DEBUG\_USART\_BOUND \* 16);  div\_mantissa = (u16)usart\_div; //得到整数部分  div\_fraction = (usart\_div - div\_mantissa) \* 16 + 0.5;//四舍五入  USART1->BRR = (div\_mantissa << 4) + div\_fraction; //设置波特率    USART1->CR1 = 0;//清空所有配置    USART1->CR1 |= (0X1 << 3);//发送器使能  USART1->CR1 |= (0X1 << 2);//接收器使能    USART1->CR1 |= (0X1 << 13);//使能USART1    USART1->CR2 &= ~(0X3 << 12);//停止位为1个位    uint32\_t priority = 0; //保存合成的优先级值  NVIC\_SetPriorityGrouping(7 - 2);  priority = NVIC\_EncodePriority(7 - 2, 2, 2);  NVIC\_SetPriority(USART1\_IRQn, priority); //以USART1为例  //根据外设寄存器配置----需要掌握该外设相关的寄存器  USART1->CR1 |= (0X1 << 5);//接收中断使能  NVIC\_EnableIRQ(USART1\_IRQn); //以USART1为例  }  /\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数功能: USART1的中断服务函数  \* 函数形参: None  \* 函数返回值: None  \* 备注: 需要清中断标志位，中断服务函数只有一个，中断进入有N个，需要判断具体的中断类型  \* 作者: LeeSong  \* 时间: 2019-08-06  \* 修改作者: None  \* 修改时间: None  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  void USART1\_IRQHandler(void)  {  uint8\_t receive\_data = 0;    //如果接收到数据，产生接收标志位  if(USART1->SR & (0x1 << 5))  {  USART1->SR &= ~(0x1 << 5); //清除标志位    //读取接收缓冲区中的数据  receive\_data = USART1->DR;    dubug\_usart\_write\_byte\_data(receive\_data); //将接收到的字节数据发送出去  }  } |

## STM32F407VG外部中断

### STM32F407VG外部中断简介

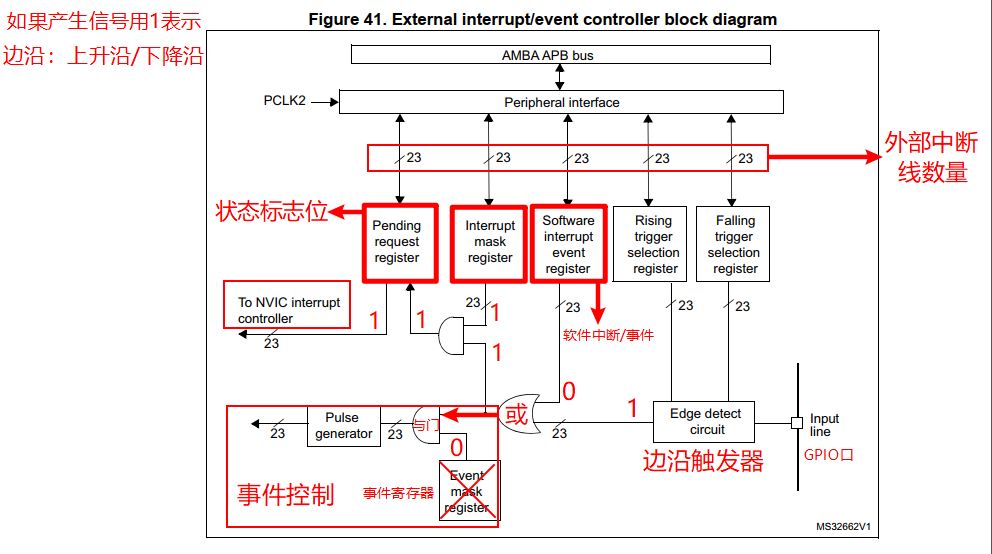
来自于IO口的中断称为外部中断。

概念补充：事件/中断。

中断：必须CPU参与，中断有中断服务函数，需要运行程序代码。

事件：不需要CPU参与，满足条件就会触发对应的标志或硬件操作。

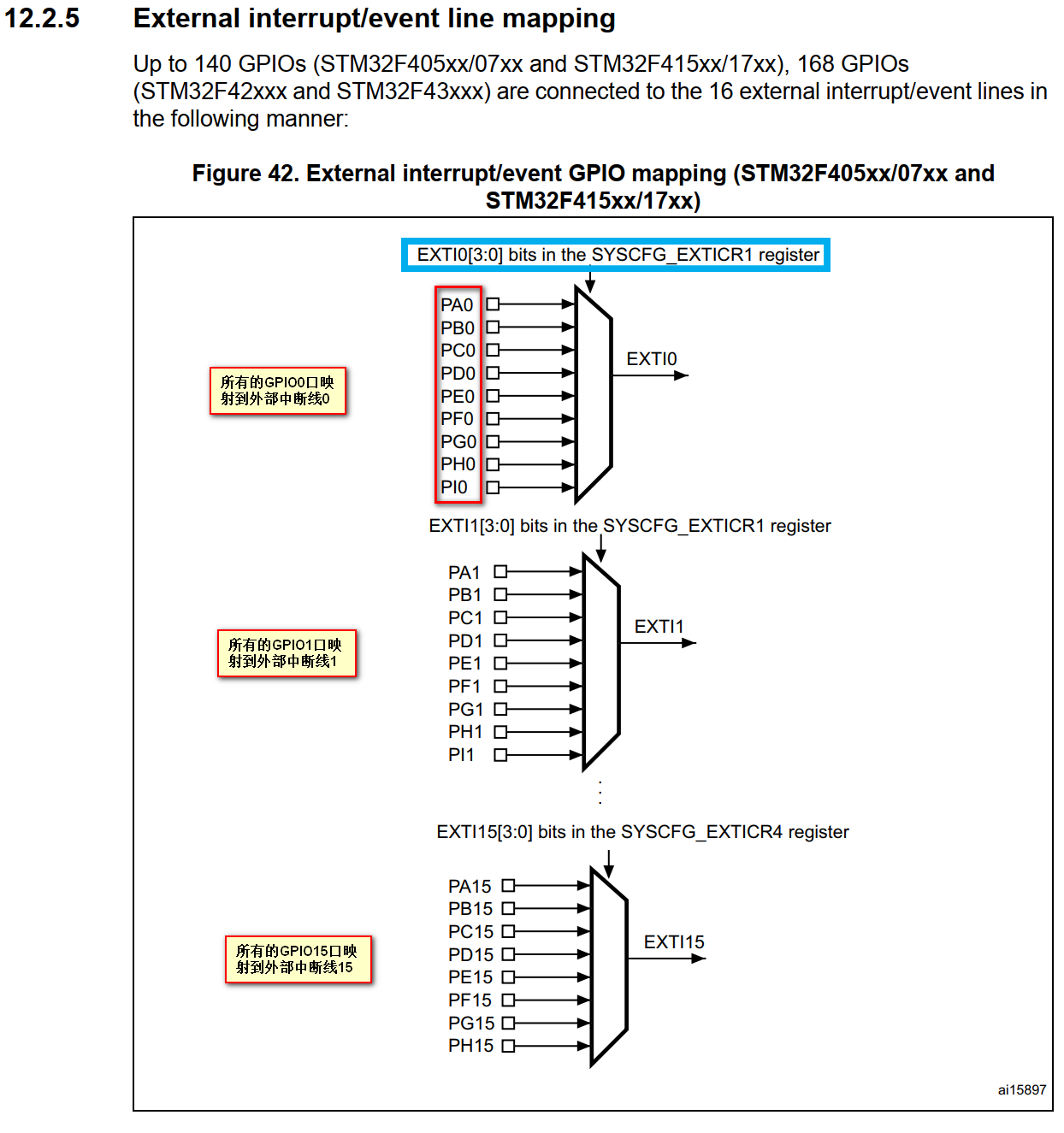
### STM32F407VG外部中断原理

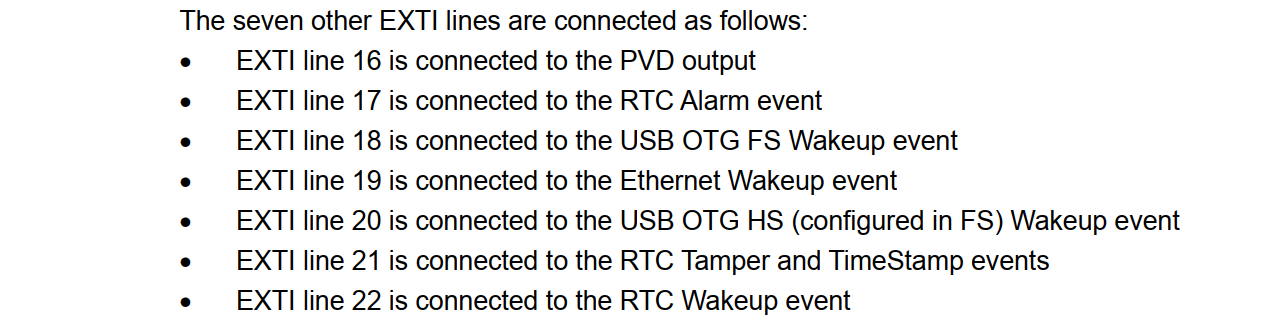


想要产生外部中断，配置步骤如下：

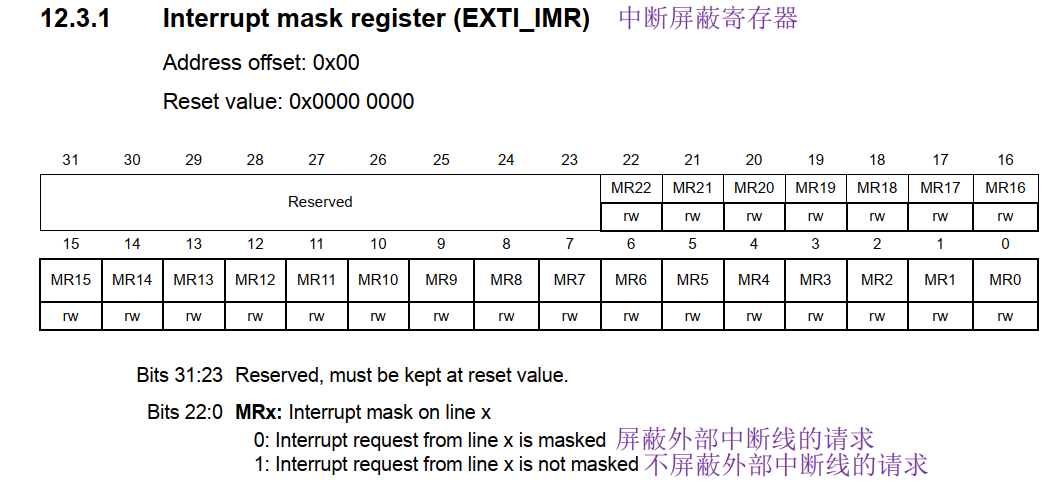
1. 选择边沿触发器(可以选上升沿，可以选下降沿，可以选择两者都触发)
2. 关闭软件中断
3. 关闭事件功能
4. 开启中断屏蔽
5. 配置NVIC控制器
   1. 设置优先级
   2. 设置NVIC中断使能

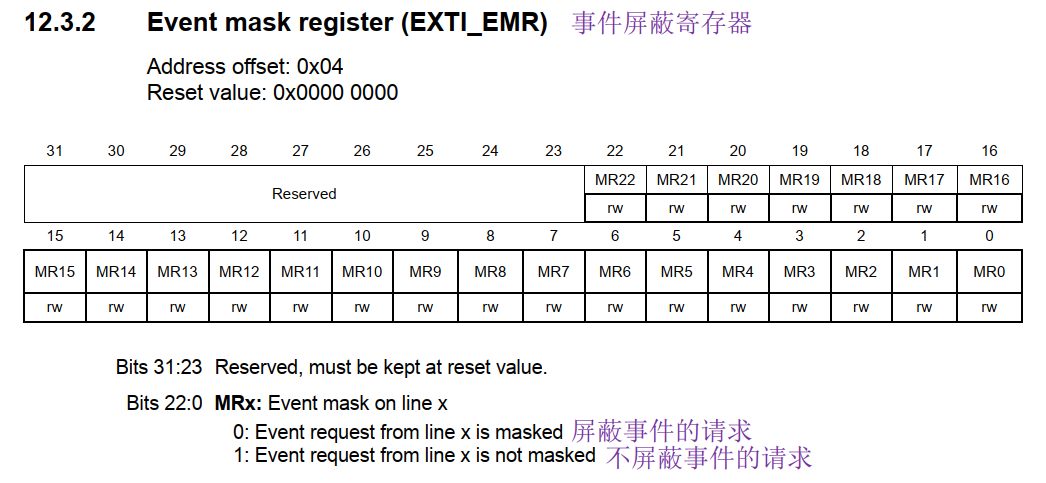
外部中断线总共有23条，其中十六条来自外部IO口，其它七条来自芯片内部外设。

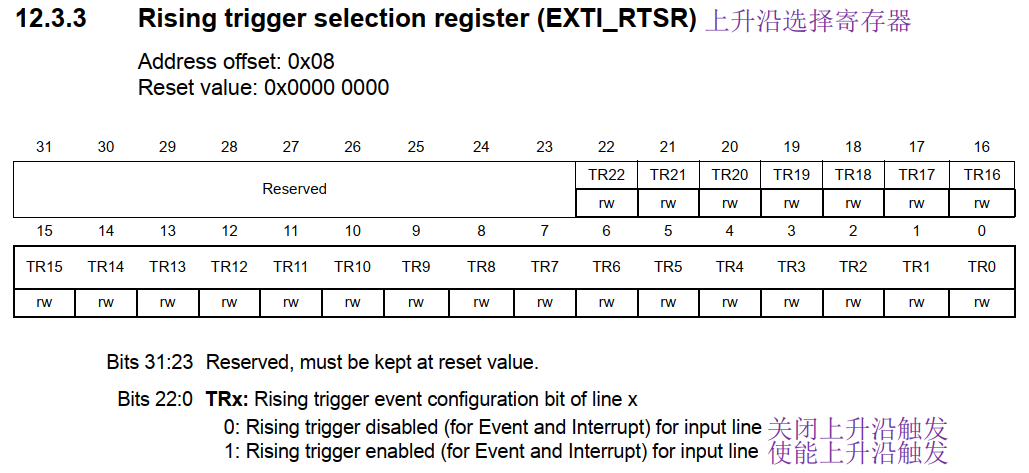


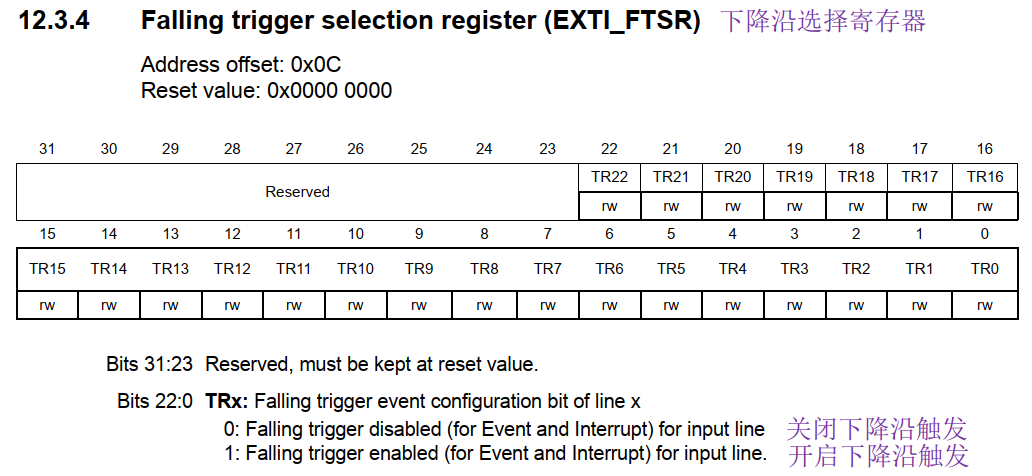


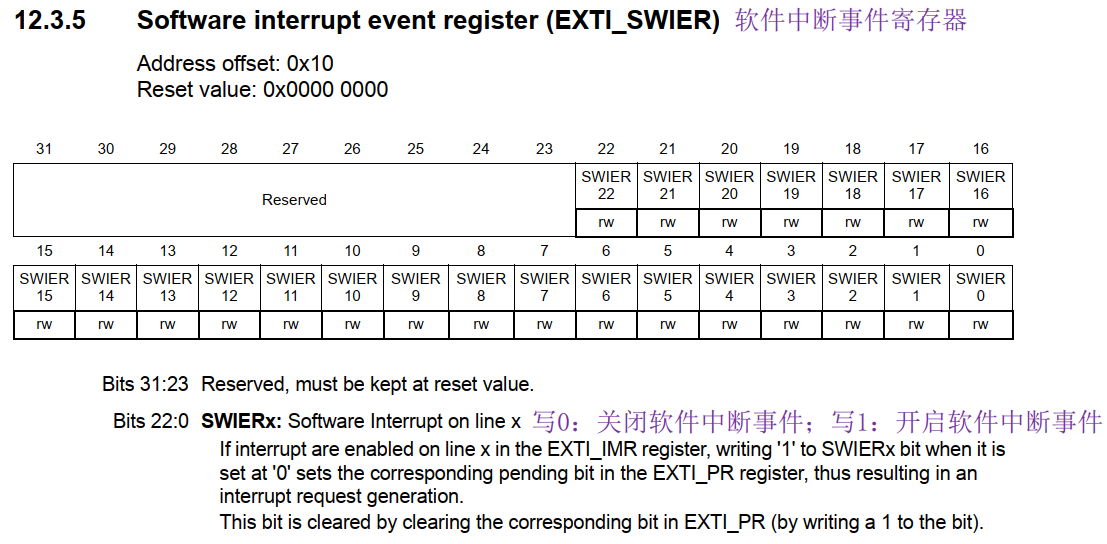
### STM32F407VG外部中断相关寄存器

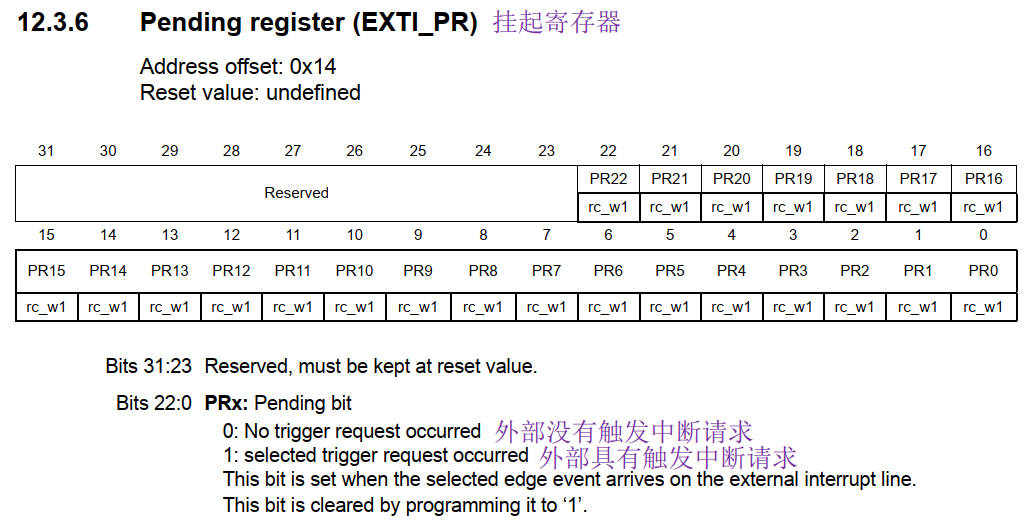


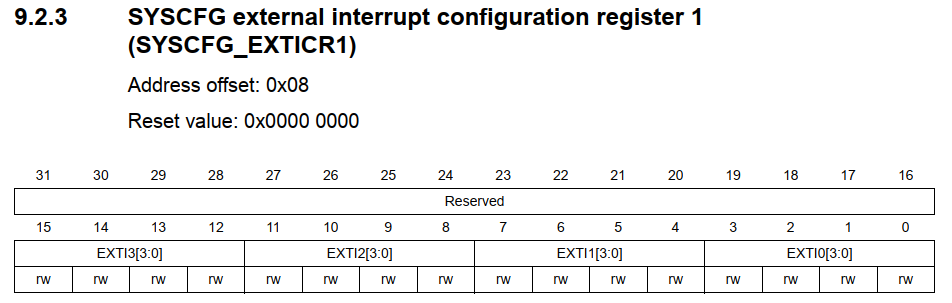


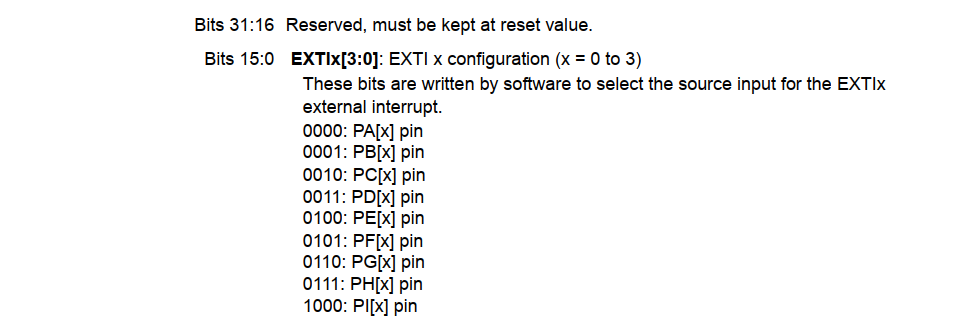


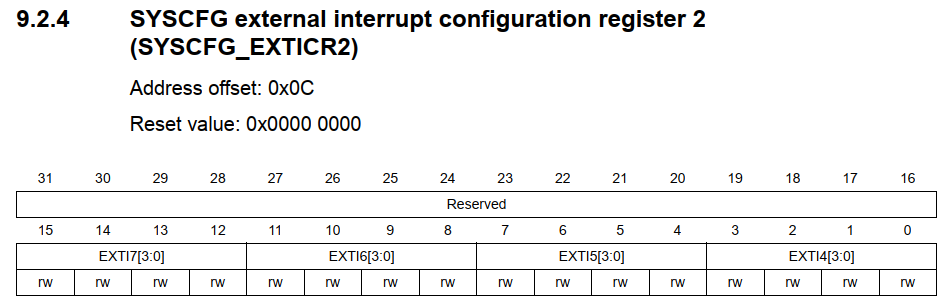


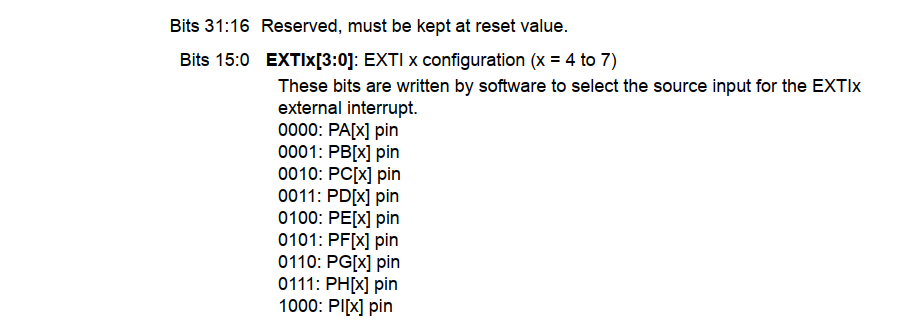


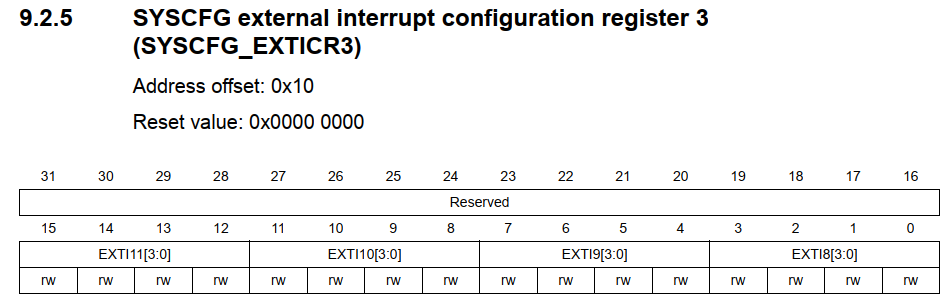


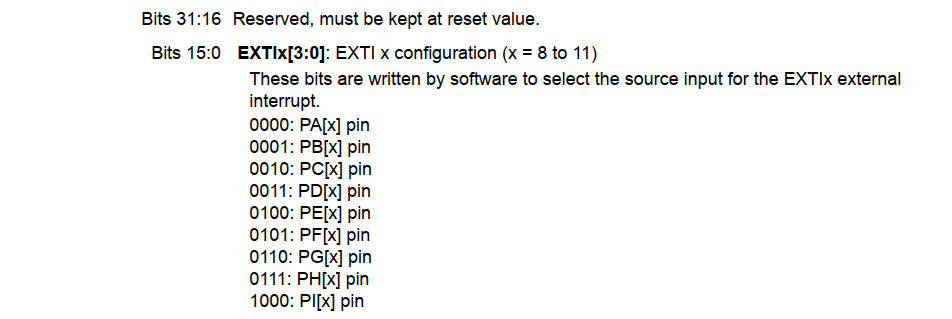


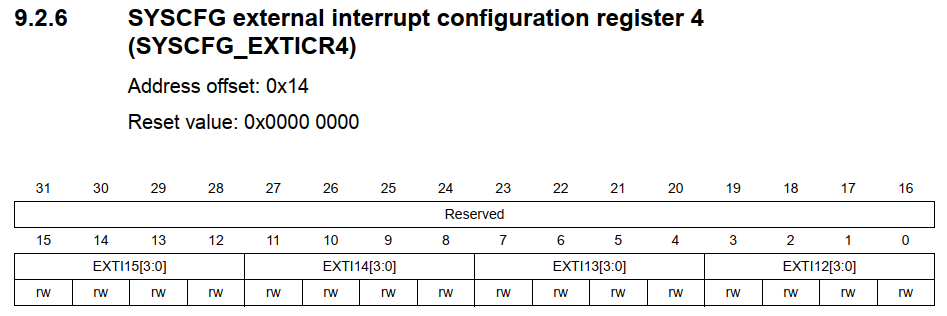


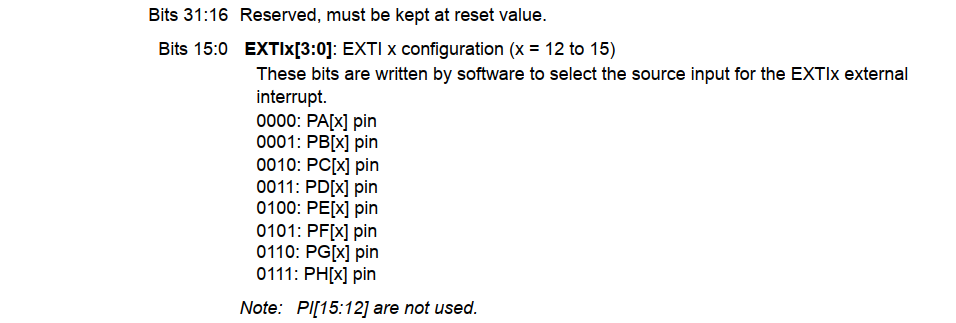


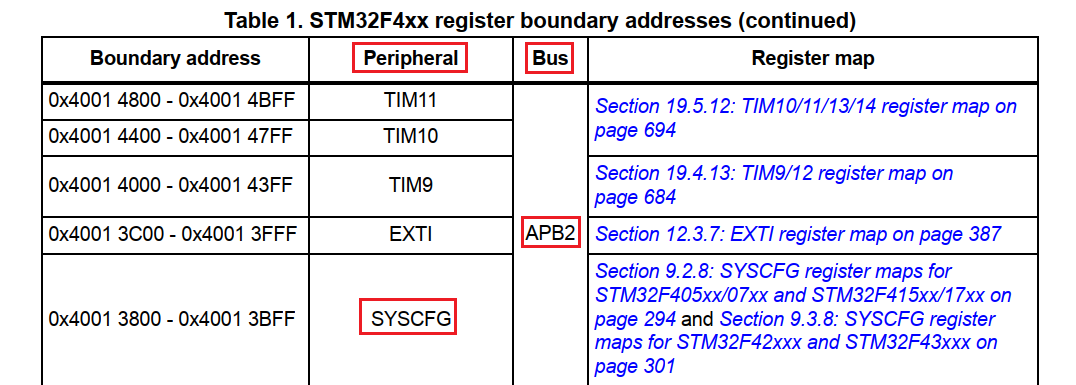


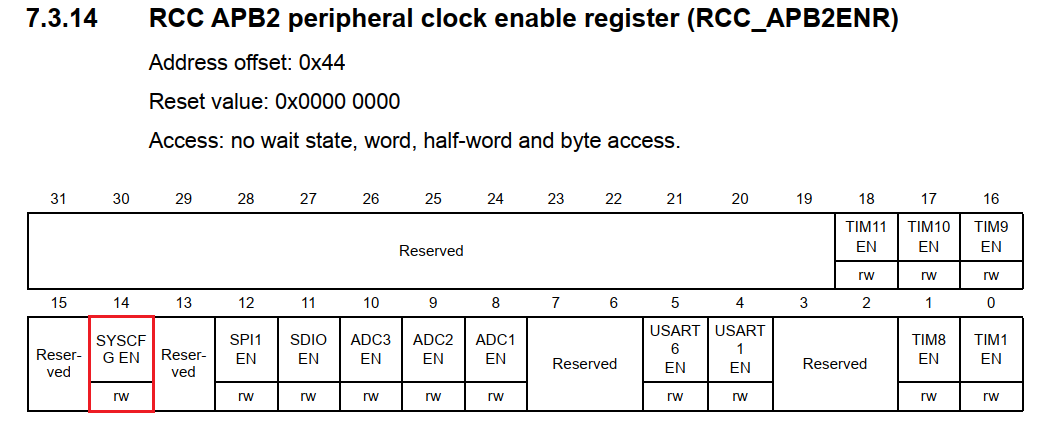






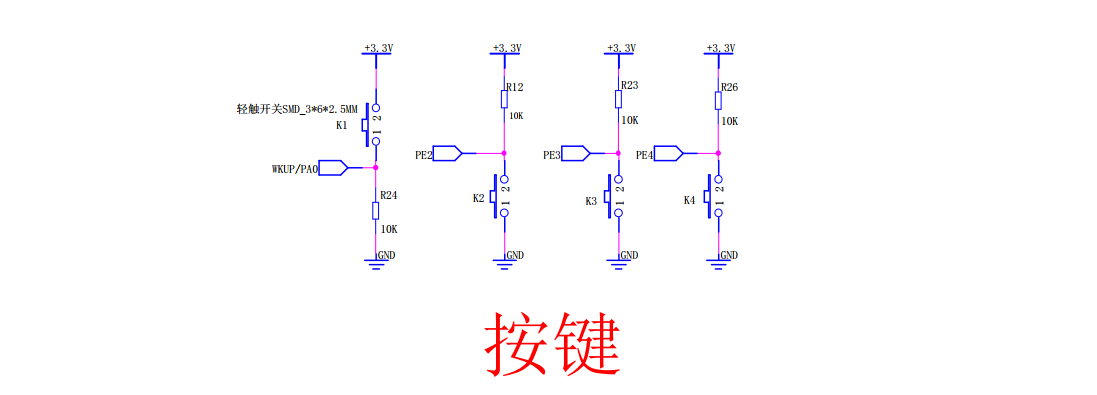






### STM32F407VG外部中断实验

利用按键触发外部中断。



出现下降沿表示按键按下。

程序设计步骤：

1. 初始化
2. 初始化GPIO口
   1. 开启时钟
   2. 配置模式为普通输入
   3. 配置为无上拉/下拉
3. 配置外部中断
   1. 开启系统配置控制器时钟
   2. 配置外部中断线对应外部IO口
   3. 配置边沿触发方式
   4. ……
4. 配置NVIC控制器
   1. 设置优先级
   2. 设置NVIC中断使能
5. 中断服务函数
6. 判断中断标志位
7. 清除中断标志位
8. 编写你需要执行的程序代码(你想干嘛就干嘛；不能长延时，程序能快就快)

## STM32F407VG软件中断

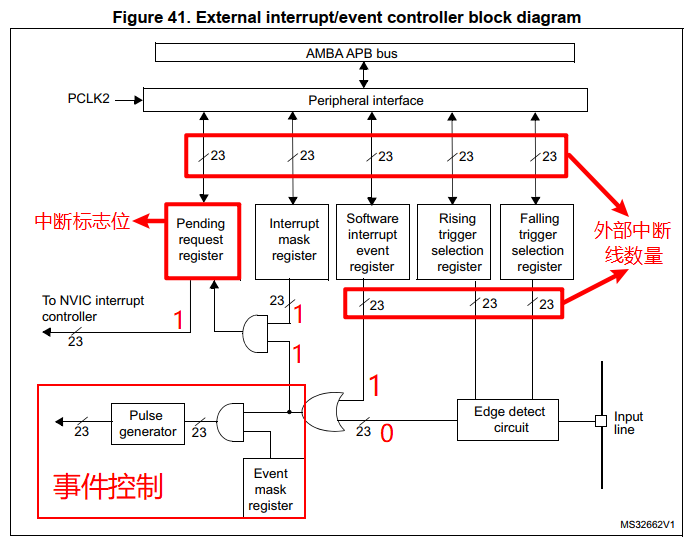
### STM32F407VG软件中断介绍

软件中断的作用：用于保护程序代码。

示例：

|  |
| --- |
| int value = 10; //value----表示USB集线器上可以使用的USB口的数量  value++; //USB集线器上的一个USB设备已经拔出可以供其他人使用；每拔出一个，value值加1  value++程序代码执行过程有几步？三步：  第一步：读取value的值  第二步：value的值加一  第三步：将value增加以后的值写入value中  程序在运行过程中，很有可能在这三步过程中被其它中断打断从而影响程序的运行，为了避免这个影响，利用软件中断对value++这行程序代码进行保护。  想要保护程序代码的操作：  将value++这行代码放到软件中断服务函数中运行  想要运行value++这行代码，只需要产生一个软件中断即可 |

### STM32F407VG软件中断工作原理



想要软件中断工作，配置步骤：

1. 关闭软件中断
2. 关闭事件功能
3. 开启中断屏蔽
4. 配置NVIC控制器
   1. 设置优先级
   2. 设置NVIC中断使能

### STM32F407VG软件中断实验

软件中断操作步骤：

1. 初始化
   1. 关闭边沿触发器
   2. 关闭软件中断
   3. 关闭事件屏蔽
   4. 开启中断屏蔽
   5. 配置NVIC控制器

注意：中断线有23条，这23条可以随意使用，需要整个项目不会用到同一条即可。

1. 软件中断服务函数

注意：软件中断服务函数和外部中断服务函数是同一个。

|  |
| --- |
| /\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数功能: 初始化软件中断  \* 函数形参: None  \* 函数返回值: None  \* 备注: None  \* 作者: LeeSong  \* 时间: 2019-08-06  \* 修改作者: None  \* 修改时间: None  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  void software\_exti\_init(void)  {  EXTI->RTSR &= ~(0x1 << 5); //关闭中断线5上升沿触发  EXTI->FTSR &= ~(0x1 << 5); //关闭中断线5下降沿触发    EXTI->SWIER &= ~(0x1 << 5); //关闭中断线5软件中断事件    EXTI->EMR &= ~(0x1 << 5); //关闭中断线5事件屏蔽    EXTI->IMR |= 0x1 << 5; //开启中断线5中断屏蔽    uint32\_t priority = 0; //保存合成的优先级值  NVIC\_SetPriorityGrouping(7 - 2);  priority = NVIC\_EncodePriority(7 - 2, 0, 0);  NVIC\_SetPriority(EXTI9\_5\_IRQn, priority); //外部中断线5  NVIC\_EnableIRQ(EXTI9\_5\_IRQn); //外部中断线5  }  /\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数功能: 外部中断线5~9的中断服务函数  \* 函数形参: None  \* 函数返回值: None  \* 备注: None  \* 作者: LeeSong  \* 时间: 2019-08-06  \* 修改作者: None  \* 修改时间: None  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  void EXTI9\_5\_IRQHandler(void)  {  if(EXTI->PR & (0x1 << 5))  {  EXTI->PR |= 0x1 << 5; //清标志位    GPIOC->ODR ^= 0x1 << 4;  }  } |